

Package ‘PSM’

May 12, 2018

Type Package

Title Non-Linear Mixed-Effects Modelling using Stochastic Differential Equations

Version 0.8-12

Date 2018-11-05

Encoding UTF-8

Author Stig Bousgaard Mortensen, Soeren Klim, Robert Miller

Maintainer Robert Miller <robert.miller@tu-dresden.de>

Imports deSolve, graphics, MASS, numDeriv, stats, ucminf

Suggests datasets

Description Functions for fitting linear and non-linear mixed-effects models using stochastic differential equations (SDEs). The package allows for any multivariate non-linear time-variant model to be specified, and it also handles multidimensional input, covariates, missing observations, and specification of dosage regimen. The provided pipeline relies on the coupling of the FOCE algorithm and Kalman filtering as outlined by Klim et al (2009, <doi:10.1016/j.cmpb.2009.02.001>) and has been validated against the proprietary software 'NONMEM' (Tornøe et al, 2005, <doi:10.1007/s11095-005-5269-5>). Further functions are provided for finding smoothed estimates of model states and for simulation.

License GPL (>= 2)

URL <http://www.imm.dtu.dk/psm>

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-05-12 16:27:48 UTC

R topics documented:

PSM-package	2
matexp	3
PSM.estimate	6
PSM.plot	11

PSM.simulate	12
PSM.smooth	14
PSM.template	17

Index	19
--------------	-----------

PSM-package	<i>Population Stochastic Modelling</i>
-------------	--

Description

Non-Linear Mixed-Effects Modelling using Stochastic Differential Equations

Functions for fitting linear and non-linear mixed-effects models using stochastic differential equations (SDEs). The package allows for any multivariate non-linear time-variant model to be specified, and it also handles multidimensional input, covariates, missing observations, and specification of dosage regimen. The provided pipeline relies on the coupling of the FOCE algorithm and Kalman filtering as outlined by Klim et al. (2009) and has been validated against the proprietary software 'NONMEM' (Tornøe et al., 2005). Further functions are provided for finding smoothed estimates of model states and for simulation.

Details

Function overview:

PSM.estimate

Estimate population parameters for any linear or non-linear model.

PSM.smooth

Optimal estimates of model states based on estimated parameters.

PSM.simulate

Simulate data for multiple individuals.

PSM.plot

Plot data, state estimates ect. for multiple individuals.

PSM.template

Creates a template with R-syntax to help setup a model in PSM.

Note

For further details please also read the package vignette pdf-document by writing `vignette("PSM")` in R.

Author(s)

Stig B. Mortensen <stigbm@gmail.com>, Soeren Klim <soren@klimens.dk>, and Robert Miller <robert.miller@tu-dresden.de>

References

Klim, S., Mortensen, S. B., Kristensen, N. R., Overgaard, R. V., & Madsen, H. (2009). Population stochastic modelling (PSM)—an R package for mixed-effects models based on stochastic differential equations. *Computer Methods and Programs in Biomedicine*, 94:279-289.

Moler, C., & Van Loan, C. (2003). Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, 45:3-49.

Mortensen, S. B., Klim, S., Dammann, B., Kristensen, N. R., Madsen, H., Overgaard, R. V. (2007). A matlab framework for estimation of NLME models using stochastic differential equations: Application for estimation of insulin secretion rates. *Journal of Pharmacokinetics and Pharmacodynamics*, 34:623-642.

Tornøe, C. W., Overgaard, R. V., Agersøe, H., Nielsen, H. A., Madsen, H., & Jonsson, E. N. (2005). Stochastic differential equations in NONMEM: implementation, application, and comparison with ordinary differential equations. *Pharmaceutical Research*, 22:1247-1258.

Web: <http://www.imm.dtu.dk/psm>

See Also

[PSM.estimate](#), [PSM.smooth](#), [PSM.simulate](#), [PSM.plot](#), [PSM.template](#)

matexp

Matrix exponential

Description

Matrix exponential of a square matrix computed by the pade approximation.

Usage

```
matexp(a, dt = 1, order = 8)
```

Arguments

a	A square numeric matrix
dt	Integration Time step
order	Pade approximation order

Details

This implementation is based on Niels Rode Kristensens work. This package is also highly inspired by David Firth's R package mexp.

Value

The matrix exponential is returned. The function issues an error if problems occurred in the fortran engine.

Note

For indepth material on matrix exponentials - see Moler and van Loan (2003).

Author(s)

Soeren Klim, Stig B. Mortensen

References

This implementation is based on Niels Rode Kristensens work. This package is also highly inspired by David Firth's R package mexp.

The examples below are all from David Firth's mexp package but the accuracy example has been removed as this package does not calculate the accuracy.

Niels Rode Kristensen, <http://www2.imm.dtu.dk/~ctsm/>

Examples

```
## The test cases have been taken directly from David Firths MEXP package.
```

```
##
```

```
##
```

```
## -----
```

```
## Test case 1 from Ward (1977)
```

```
## -----
```

```
test1 <- t(matrix(c(
  4, 2, 0,
  1, 4, 1,
  1, 1, 4), 3, 3))
```

```
matexp(test1)
```

```
## Results on Power Mac G3 under Mac OS 10.2.8
```

```
##           [,1]           [,2]           [,3]
```

```
## [1,] 147.86662244637000 183.76513864636857 71.79703239999643
```

```
## [2,] 127.78108552318250 183.76513864636877 91.88256932318409
```

```
## [3,] 127.78108552318204 163.67960172318047 111.96810624637124
```

```
## -- these agree with ward (1977, p608)
```

```
##
```

```
## A naive alternative to mexp, using spectral decomposition:
```

```
mexp2 <- function(matrix){
  z <- eigen(matrix,sym=FALSE)
  Re(z$vectors %*% diag(exp(z$values)) %*%
    solve(z$vectors))
}
```

```
}
```

```
try(mexp2(test1))
```

```
## now gives an error from solve !
```

```
##
```

```
## older result was
```

```
##           [,1]           [,2]           [,3]
```

```
##[1,] 147.86662244637003 88.500223574029647 103.39983337000028
```

```

##[2,] 127.78108552318220 117.345806155250600 90.70416537273444
##[3,] 127.78108552318226 90.384173332156763 117.66579819582827
## -- hopelessly inaccurate in all but the first column.
##
##
## -----
## Test case 2 from Ward (1977)
## -----
test2 <- t(matrix(c(
  29.87942128909879, .7815750847907159, -2.289519314033932,
  .7815750847907159, 25.72656945571064, 8.680737820540137,
  -2.289519314033932, 8.680737820540137, 34.39400925519054),
  3, 3))

matexp(test2)

##           [,1]           [,2]           [,3]
##[1,]  5496313853692357 -18231880972009844 -30475770808580828
##[2,] -18231880972009852  60605228702227024 101291842930256144
##[3,] -30475770808580840 101291842930256144 169294411240859072
## -- which agrees with Ward (1977) to 13 significant figures
mexp2(test2)
##           [,1]           [,2]           [,3]
##[1,]  5496313853692405 -18231880972009100 -30475770808580196
##[2,] -18231880972009160  60605228702221760 101291842930249376
##[3,] -30475770808580244 101291842930249200 169294411240850880
## -- in this case a very similar degree of accuracy.
##
## -----
## Test case 3 from Ward (1977)
## -----
test3 <- t(matrix(c(
  -131, 19, 18,
  -390, 56, 54,
  -387, 57, 52), 3, 3))

matexp(test3)

##           [,1]           [,2]           [,3]
##[1,] -1.5096441587713636 0.36787943910439874 0.13533528117301735
##[2,] -5.6325707997970271 1.47151775847745725 0.40600584351567010
##[3,] -4.9349383260294299 1.10363831731417195 0.54134112675653534
## -- agrees to 10dp with Ward (1977), p608.

mexp2(test3)

##           [,1]           [,2]           [,3]
##[1,] -1.509644158796182 0.3678794391103086 0.13533528117547022
##[2,] -5.632570799902948 1.4715177585023838 0.40600584352641989
##[3,] -4.934938326098410 1.1036383173309319 0.54134112676302582
## -- in this case, a similar level of agreement with Ward (1977).
##

```

PSM.estimate

*Estimate population parameters***Description**

Estimates population parameters in a linear or non-linear mixed effects model based on stochastic differential equations by use of maximum likelihood and the Kalman filter.

Usage

```
PSM.estimate(Model, Data, Par, CI = FALSE, trace = 0, control=NULL, fast=TRUE)
```

Arguments

Model

A list containing the following elements:

Matrices = function(phi) *Only in linear models.*

Defines the matrices A , B , C and D in the model equation. Must return a list of matrices named matA, matB, matC and matD. If there is no input, matB and matD may be omitted by setting them to NULL. Note, if the matrix A is singular the option fast is set to FALSE, as this is not supported in the compiled Fortran code.

Functions *Only in non-linear models.*

A list containing the functions $f(x, u, time, phi)$, $g(x, u, time, phi)$, $df(x, u, time, phi)$ and $dg(x, u, time, phi)$.

The functions f and g defines the system and df and dg are the Jacobian matrices with first-order partial derivatives for $f(x)$ and $g(x)$ which is needed to evaluate the model. A warning is issued if df or dg appear to be incorrect based on a numerical evaluation of the Jacobians of $f(x)$ and $g(x)$.

It is possible to avoid specifying the Jacobian functions in the model and use numerical approximations instead, but this will increase estimation time at least ten-fold. See the section 'Numerical Jacobians of f and g ' below for more information.

$X0$ = function(Time, phi, U) Defines the model state at Time[1] before update. Time[1] and U[,1] can be used in the evaluation of $X0$. Must return a column matrix.

SIG = function(phi) in linear models and SIG = function(u, time, phi) in non-linear models. It defines the matrix σ for the diffusion term. Returns a square matrix.

S = function(phi) in linear models and S = function(u, time, phi) in non-linear models. It defines a covariance matrix for the observation noise. Returns a square matrix.

h = function(eta, theta, covar) Second stage model. Defines how random effects (eta) and covariates (covar) affects the fixed effects parameters (theta). In models where OMEGA=NULL (no random-effects) h must still be defined with the same argument list to allow for covariates to affect theta, but the function h is evaluated with eta=NULL. Must return a list (or vector)

	<p>phi of individual parameters which is used as input argument in the other user-defined functions.</p> <p>ModelPar = function(THETA) Defines the population parameters to be optimized. Returns a list containing 2 elements, named:</p> <p>theta A list of fixed effects parameters θ which are used as input to the function h listed above.</p> <p>OMEGA A square covariance matrix Ω for the random effects. If OMEGA is missing or NULL then no 2nd stage model is used. However, the function h must still be defined, see above.</p>
Data	<p>An unnamed list where each element contains data for one individual. Each element in Data is a list containing:</p> <p>Time A vector of timepoints for measurements</p> <p>Y A matrix of multivariate observations for each timepoint, where each column is a multivariate measurement. Y may contain NA for missing observations and a column may consist of both some or only NAs. The latter is useful if a dose is given when no measurement is taken.</p> <p>U A matrix of multivariate input to the model for each timepoint. U is assumed constant between measurements and may not contain any NA. If U is omitted, the model is assumed to have no input and matB and matD need no to be specified.</p> <p>Dose A list containing the 3 elements listed below. If the element Dose is missing or NULL, no dose is assumed.</p> <p>Time A vector of timepoints for the dosing. Each must coincide with a measurement time. Remember to insert a missing measurement in Y if a corresponding timepoint is not present. Dose is considered added to the system just after the measurement.</p> <p>State A vector with indexes of the state for dosing.</p> <p>Amount A vector of amounts to be added.</p>
Par	<p>A list containing the following elements:</p> <p>Init A vector with initial estimates for THETA, vector of population parameters to be optimized.</p> <p>LB, UB : Two vectors with lower and upper bounds for parameters. If omitted, the program performs unconstrained optimization. It is highly recommended to specify bounds to ensure robust optimization.</p>
CI	<p>Boolean. If true, the program estimates 95% confidence intervals, standard deviation and correlation matrix for the parameter estimates based on the Hessian of the likelihood function. The Hessian is estimated by hessian in the numDeriv package.</p>
trace	<p>Non-negative integer. If positive, tracing information on the progress of the optimization is produced. Higher values produces more tracing information.</p>
control	<p>A list of control parameters for the optimization of the likelihood function. The list has one required component, namely:</p> <p>optimizer A string value equal to either 'optim' or 'ucminf'. This gives the choice of optimizer. Default is optimizer = 'optim'.</p>

The remaining components in the list are given as the control argument for the chosen optimizer. See corresponding help file for further detail.

fast Boolean. Use compiled Fortran code for faster estimation.

Details

The first stage model describing intra-individual variations is for linear models defined as

$$dx_t = (A(\phi_i)x_t + B(\phi_i)u_t)dt + \sigma(\phi_i)d\omega_t$$

$$y_{ij} = C(\phi_i)x_{ij} + D(\phi_i)u_{ij} + e_{ij}$$

and for non-linear models as

$$dx_t = f(x_t, u_t, t, \phi_i)dt + \sigma(u_t, t, \phi_i)d\omega_t$$

$$y_{ij} = g(x_{ij}, u_{ij}, t_{ij}, \phi_i) + e_{ij}$$

where $e_{ij} \sim N(0, S(u_{ij}, t_{ij}, \phi_i))$ and ω_t is a standard Brownian motion.

The second stage model describing inter-individual variations is defined as:

$$\phi_i = h(\eta_i, \theta, Z_i)$$

where $\eta_i \sim N(0, \Omega)$, θ are the fixed effect parameters and Z_i are covariates for individual i . In a model without random-effects the function h is only used to include possible covariates in the model.

Value

A list containing the following elements:

NegLogL	Value of the negative log-likelihood function at optimum.
THETA	Population parameters at optimum
CI	95% confidence interval for the estimated parameters
SD	Standard deviation for the estimated parameters
COR	Correlation matrix for the estimated parameters
sec	Time for the estimation in seconds
opt	Raw output from <code>optim</code>

Numerical Jacobians of f and g

Automatic numerical approximations of the Jacobians of f and g can be used in PSM. In the following, the name of the model object is assumed to be `MyModel`.

First define the functions `MyModel$Functions$f` and `MyModel$Functions$g`. When these are defined in `MyModel` the functions `df` and `dg` can be added to the model object by writing as below:


```
MyModel$Functions$df = function(x,u,time,phi) {  
  jacobian(MyModel$Functions$f,x=x,u=u,time=time,phi=phi)  
}  
MyModel$Functions$dg = function(x,u,time,phi) {  
  jacobian(MyModel$Functions$g,x=x,u=u,time=time,phi=phi)  
}
```

This way of defining df and dg forces a numerical evaluation of the Jacobians using the **numDeriv** package. It may be useful in some cases, but it should be stressed that it will probably give at least a ten-fold increase in estimation times.

Note

For further details please also read the package vignette pdf-document by writing `vignette("PSM")` in R.

Author(s)

Stig B. Mortensen and Soeren Klim

References

Please visit <http://www.imm.dtu.dk/psm> or refer to the main help page for [PSM](#).

See Also

[PSM](#), [PSM.smooth](#), [PSM.simulate](#), [PSM.plot](#), [PSM.template](#)

Examples

```
#detailed examples are provided in the package vignette  
  
#Theophylline data from Boeckmann et al (1994)  
#objective: recover the administered doses  
  
library(datasets)  
data(Theoph)  
  
#reshape data to PSM format  
  
TheophPSM = list()  
for(i in 1:length(unique(Theoph$Subject))){  
  TheophPSM[[i]] = with(  
    Theoph[Theoph$Subject == i,],  
    list(Y = matrix(conc, nrow=1), Time = Time)  
  )  
}
```

```

#specify a simple pharmacokinetic model comprised of
#2 state equations and 1 observation equation
#initial value of 1 state eq. varies randomly across individuals

mod = vector(mode="list")
mod$Matrices = function(phi) {
  list(
    matA=matrix(c(-phi$ka, 0, phi$ka, -phi$ke), nrow=2, ncol=2, byrow=TRUE),
    matC=matrix(c(0, 1), nrow=1, ncol=2)
  )
}
mod$h = function(eta, theta, covar) {
  phi = theta
  phi$dose = theta$dose * exp(eta[1])
  phi
}
mod$S = function(phi) {
  matrix(c(phi$sigma), nrow=1, ncol=1)
}
mod$SIG = function(phi) {
  matrix(c(0, 0, 0, phi$omega), nrow=2, ncol=2, byrow=TRUE)
}
mod$X0 = function(Time, phi, U) {
  matrix(c(phi$dose, 0), nrow=2, ncol=1)
}
mod$ModelPar = function(THETA) {
  list(theta=list(dose = THETA["dose"], ka = THETA["ka"], ke = THETA["ke"],
    omega = THETA["omega"], sigma = THETA["sigma"]),
    OMEGA=matrix(c(THETA["BSV_dose"]), nrow=1, ncol=1)
  )
}

#specify the search space of the fitting algorithm

parM = c(ka = 1.5, ke = 0.1, dose = 10, omega = .3, sigma = 1,
  BSV_dose = 0.015)
pars = list(LB=parM*.25, Init=parM, UB=parM*1.75)

#fit model and predict data

fit = PSM.estimate(mod, TheophPSM, pars, trace = 1, fast = TRUE,
  control=list(optimizer="optim", maxit=1))
pred = PSM.smooth(mod, TheophPSM, fit$THETA)

#visualize recovery performance

true_dose = tapply(Theoph$conc, Theoph$Subject, mean)
true_dose = true_dose[order(as.numeric(names(true_dose)))]
est_dose = fit$THETA["dose"] * exp(unlist(lapply(pred, function(x) x$eta)))
plot(true_dose, est_dose,
  xlab="actually administered dose", ylab="recovered dose")

```

```
abline(lm(est_dose ~ true_dose), lty=2)
```

 PSM.plot

Basic plots of data and output

Description

Create basic plots of data and state estimates in PSM.

Usage

```
PSM.plot(Data, Smooth = NULL, indiv = NULL, type = NULL)
```

Arguments

Data	Data list, see description in PSM.estimate .
Smooth	Output from PSM.smooth .
indiv	A vector of integers with which individuals to include.
type	A vector of strings listing the types of plots to create. The possibilities are: <ul style="list-style-type: none"> ‘Y’ Observations ‘U’ Input ‘X’ Simulated states at sample times ‘longX’ Simulated states with time increment <code>deltaTime</code> ‘Xp’ Predicted states ‘Xf’ Filtered states ‘Xs’ Smoothed states ‘Yp’ Response based on predicted state ‘Ys’ Response based on smoothed state ‘Yp.Y’ As above with observations added ‘Ys.Y’ As above with observations added ‘res’ Residuals (Y-Yp) ‘acf’ Auto-correlation of residuals ‘eta’ Shows estimates of random effects in plot. If Smooth is not given it will show the value of simulated random effects if they are contained in Data. If a string is preceded by ‘logx.’, ‘logy.’ or ‘logx.logy.’ the corresponding axis is shown on log-scale. An example is: <code>type=c('Xs', 'logy.Ys.Y', 'res', 'eta')</code>

Value

None (invisible NULL).

Note

For further details please also read the package vignette pdf-document by writing `vignette("PSM")` in R.

Author(s)

Stig B. Mortensen and Soeren Klim

References

Please visit <http://www.imm.dtu.dk/psm> or refer to the help page for [PSM](#).

See Also

[PSM](#), [PSM.estimate](#), [PSM.smooth](#), [PSM.simulate](#), [PSM.template](#)

Examples

```
cat("\nExamples are included in the package vignette.\n")
```

PSM.simulate

Create simulation data for multiple individuals

Description

Simulates data for multiple individuals in a mixed effects model based on stochastic differential equations using an euler scheme.

Usage

```
PSM.simulate(Model, Data, THETA, deltaTime, longX=TRUE)
```

Arguments

Model	A list containing the model components either Linear or Non-Linear Model list.*
Data	List with elements described below. No Data\$Y is needed as it is generated through the simulation. The number of individuals simulated is equal to length(Data). Time Time vector U Input list for the Model covar Covariates list
THETA	Vector of population parameters
deltaTime	Time Step in the Euler scheme
longX	Boolean. Toggles output of the entire simulated outcome of the states * See description in PSM.estimate .

Details

The η_i is drawn from the multivariate normal distribution $N(0, \Omega)$. The simulation is an euler based method but for every time interval dt the model is predicted and the states affected by system noise (σ).

The measurements are added an normal error term belonging to $N(0, S)$.

The function `mvrnorm` from the MASS pacakge is used to to generate random numbers fra multi-variate normal distributions.

Value

The simulated outcome of the model is returned in a list, where each element is the data for an individual.

X	Simulated states sampled at time points for measurements
Y	Simulated measurements
Time	Time points for measurements
U	Input vector used in the simulation
eta	The random effects used in the simulation
Dose	The dose list used in the simulation
longX	Entire outcome of simulated states
longTime	Time points for longX.

Note

For further details please also read the package vignette pdf-document by writing `vignette("PSM")` in R.

Author(s)

Stig B. Mortensen and Soeren Klim

References

Please visit <http://www.imm.dtu.dk/psm> or refer to the help page for `PSM`.

See Also

[PSM](#), [PSM.estimate](#), [PSM.smooth](#), [PSM.plot](#), [PSM.template](#)

Examples

```
#specify pharmacokinetic model
#2 state equations, 1 observation equation, 1 random effect

mod = vector(mode="list")
mod$Matrices = function(phi) {
  list(
```

```

    matA=matrix(c(-phi$ka, 0, phi$ka, -phi$ke), nrow=2, ncol=2, byrow=TRUE),
    matC=matrix(c(0, 1), nrow=1, ncol=2)
  )
}
mod$h = function(eta, theta, covar) {
  phi = theta
  phi$dose = theta$dose * exp(eta[1])
  phi
}
mod$s = function(phi) {
  matrix(c(phi$sigma), nrow=1, ncol=1)
}
mod$SIG = function(phi) {
  matrix(c(0, 0, 0, phi$omega), nrow=2, ncol=2, byrow=TRUE)
}
mod$X0 = function(Time, phi, U) {
  matrix(c(phi$dose, 0), nrow=2, ncol=1)
}
mod$modelPar = function(THETA) {
  list(theta=list(dose = THETA["dose"], ka = THETA["ka"], ke = THETA["ke"],
                omega = THETA["omega"], sigma = THETA["sigma"]),
        OMEGA=matrix(c(THETA["BSV_dose"]), nrow=1, ncol=1)
  )
}

#specify sampling scheme and RNG

TheophPSM <- list()
TheophPSM[[1]] <- list(Time = seq(0,25,5))
set.seed(12345)

#simulate and visualize ODE model (no volatility)

parM <- c(ka = 1.58, ke = 0.08, dose = 9.54, omega = 0, sigma = 1.05, BSV_dose = 0)
TheophSim <- PSM.simulate(mod, TheophPSM, THETA = parM, deltaTime = 0.1)
plot(TheophSim[[1]]$longTime, TheophSim[[1]]$longX[2,],
     type="l", ylab="concentration", xlab="time")

#contrast it to SDE model

parM <- c(ka = 1.58, ke = 0.08, dose = 9.54, omega = 0.34, sigma = 1.05, BSV_dose = 0)
TheophSim <- PSM.simulate(mod, TheophPSM, THETA = parM, deltaTime = 0.1)
lines(TheophSim[[1]]$longTime, TheophSim[[1]]$longX[2,],
      ylab="concentration", xlab="time")

```

Description

Gives estimates of model states and random effects η . The function is intended to be used based on population parameters found using [PSM.estimate](#) or to check initial values before parameter estimation.

Usage

```
PSM.smooth(Model, Data, THETA, subsample = 0, trace = 0, etaList = NULL)
```

Arguments

Model	Model list.*
Data	Data list.*
THETA	Vector of population parameters used for the state estimation.
subsample	Number of points to estimate states in between measurements. The extra points are linearly spaced.
trace	Non-negative integer. If positive, tracing information on the progress of the optimization is produced. Higher values produces more tracing information.
etaList	Matrix where each column contains an estimate of η_i . etaList has the same format as the output of PSM.estimate . If omitted, the function will evaluate the population likelihood function to find estimates of η_i for all individuals. * See description in PSM.estimate .

Details

The function produces three types of estimates.

Predicted Only past measurements are used for the state estimate at time t.

Filtered Only past and the current measurements are used for the state estimate at time t.

Smoothed All measurements (both past and future) are used to form the state estimate at time t. This is usually the preferred type of state estimate.

If `subsample>0` then the data is automatically subsampled to provide estimated of the model states between observation time points.

Value

An unnamed list with one element for each individual. Each element contains the following elements:

Time	Possibly subsampled time-vector corresponding to the estimated states
Xs, Ps	Smoothed state and state co-variance estimate
Ys	Response based on smoothed state: $Ys = g(Xs)$.
Xf, Pf	Filtered state and state co-variance estimate
Xp, Pp	Predicted state and state co-variance estimate
Yp, R	Predicted observations and observation variances

eta	Estimated eta
etaSE	Standard errors of eta
negLogL	Value of the negative log-likelihood function at THETA (thus same value for all individuals).

Note

For further details please also read the package vignette pdf-document by writing `vignette("PSM")` in R.

Author(s)

Stig B. Mortensen, Soeren Klim, and Robert Miller

References

Please visit <http://www.imm.dtu.dk/psm> or refer to the help page for `PSM`.

See Also

`PSM`, `PSM.estimate`, `PSM.simulate`, `PSM.plot`, `PSM.template`

Examples

```
#detailed examples are provided in the package vignette

#Theophylline data from Boeckmann et al (1994)
#objective: recover the administered doses

library(datasets)
data(Theoph)

#reshape data to PSM format

TheophPSM = list()
for(i in 1:length(unique(Theoph$Subject))){
  TheophPSM[[i]] = with(
    Theoph[Theoph$Subject == i,],
    list(Y = matrix(conc, nrow=1), Time = Time)
  )
}

#specify a simple pharmacokinetic model comprised of
#2 state equations and 1 observation equation
#initial value of 1 state eq. varies randomly across individuals

mod = vector(mode="list")
mod$Matrices = function(phi) {
  list(
```



```

    matA=matrix(c(-phi$ka, 0, phi$ka, -phi$ke), nrow=2, ncol=2, byrow=TRUE),
    matC=matrix(c(0, 1), nrow=1, ncol=2)
  )
}
mod$h = function(eta, theta, covar) {
  phi = theta
  phi$dose = theta$dose * exp(eta[1])
  phi
}
mod$S = function(phi) {
  matrix(c(phi$sigma), nrow=1, ncol=1)
}
mod$SIG = function(phi) {
  matrix(c(0, 0, 0, phi$omega), nrow=2, ncol=2, byrow=TRUE)
}
mod$X0 = function(Time, phi, U) {
  matrix(c(phi$dose, 0), nrow=2, ncol=1)
}
mod$ModelPar = function(THETA) {
  list(theta=list(dose = THETA["dose"], ka = THETA["ka"], ke = THETA["ke"],
                omega = THETA["omega"], sigma = THETA["sigma"]),
        OMEGA=matrix(c(THETA["BSV_dose"]), nrow=1, ncol=1)
  )
}

#specify the search space of the fitting algorithm

parM = c(ka = 1.5, ke = 0.1, dose = 10, omega = .3, sigma = 1,
         BSV_dose = 0.015)
pars = list(LB=parM*.25, Init=parM, UB=parM*1.75)

#fit model and predict data

fit = PSM.estimate(mod, TheophPSM, pars, trace = 1, fast = TRUE,
                  control=list(optimizer="optim", maxit=1))
pred = PSM.smooth(mod, TheophPSM, fit$THETA)

#visualize recovery performance

true_dose = tapply(Theoph$conc, Theoph$Subject, mean)
true_dose = true_dose[order(as.numeric(names(true_dose)))]
est_dose = fit$THETA["dose"] * exp(unlist(lapply(pred, function(x) x$eta)))
plot(true_dose, est_dose,
     xlab="actually administered dose", ylab= "recovered dose")
abline(lm(est_dose ~ true_dose), lty=2)

```

Description

Creates a template with R-syntax to help setup a model in PSM.

Usage

```
PSM.template(Linear=FALSE,dimX=2,dimY=3,dimU=4,dimEta=5,file="")
```

Arguments

Linear	Boolean. Linear or non-linear model.
dimX	Number of state equations.
dimY	Number of response variables.
dimU	Number of input variables (can be zero).
dimEta	Number of random effects (can be zero).
file	A character string naming the file to print to. If "" (the default), PSM.template prints to the standard output connection.

Value

None (invisible NULL).

Note

For further details please also read the package vignette pdf-document by writing `vignette("PSM")` in R.

Author(s)

Stig B. Mortensen and Soeren Klim

References

Please visit <http://www.imm.dtu.dk/psm> or refer to the help page for PSM.

See Also

[PSM](#), [PSM.estimate](#), [PSM.smooth](#), [PSM.template](#)

Examples

```
# Linear model with input, random effects and dose
PSM.template(Linear=TRUE,dimX=1,dimY=2,dimU=3,dimEta=4)

# Non-linear model without input, random effects and dose
PSM.template(Linear=FALSE,dimX=1,dimY=2,dimU=0,dimEta=0)
```

Index

*Topic **htest**

PSM.estimate, 6
PSM.plot, 11
PSM.simulate, 12
PSM.smooth, 14
PSM.template, 17

*Topic **math**

matexp, 3

*Topic **models**

PSM.estimate, 6
PSM.plot, 11
PSM.simulate, 12
PSM.smooth, 14
PSM.template, 17

*Topic **multivariate**

PSM.estimate, 6
PSM.plot, 11
PSM.simulate, 12
PSM.smooth, 14
PSM.template, 17

*Topic **package**

PSM-package, 2

*Topic **ts**

PSM.estimate, 6
PSM.plot, 11
PSM.simulate, 12
PSM.smooth, 14
PSM.template, 17

matexp, 3

PSM, 9, 12, 13, 16, 18

PSM (PSM-package), 2

PSM-package, 2

PSM.estimate, 3, 6, 11–13, 15, 16, 18

PSM.plot, 3, 9, 11, 13, 16

PSM.simulate, 3, 9, 12, 12, 16

PSM.smooth, 3, 9, 11–13, 14, 18

PSM.template, 3, 9, 12, 13, 16, 17, 18